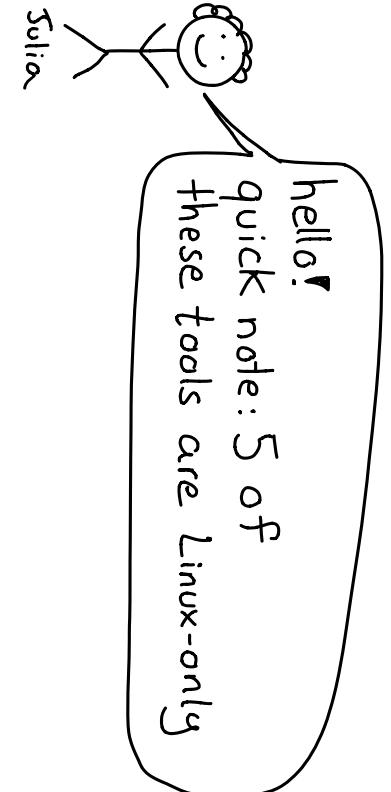


love this?  
more zines at  
→ [wizardzines.com](http://wizardzines.com) ←





## Linux only tool

## BSD/Mac equivalent

<code>ip</code>	<code>ifconfig, route</code>
<code>tc</code>	<code>dummynet (?) (BSD)</code>
<code>ss</code>	<code>netstat</code>
<code>iptables</code>	<code>pf (BSD)</code>
<code>ethtool</code>	<code>ifconfig, kind of (?)</code>

## ethtool

23

<b>--identify INTERFACE</b>	ethtool is for people who need to manage physical networks
<b>-S INTERFACE</b>	blink the light on the ethernet port. good if you have multiple ports! and cute!
<b>show statistics</b>	works like bytes sent. works for wifi interfaces too.

<b>-S</b>	ethtool <code>eth0</code> <small>name of network interface</small>
<b>change speed/duplex / other settings of an interface</b>	<code>ethtool -s eth0 speed 100</code>
<b>-i INTERFACE</b>	show firmware info

<b>iw dev wlan0 link</b>	<code>--show-offload</code> <code>--offload</code>
<b>ethtool</b>	your network card can do a lot for you! Like computing checksums. This is called "offloading". This lets you see/change configured offloads.
<b>To see the speed (and more) of a wireless connection, use iw.</b>	

# conntrack

## conntrack

not a command line tool:  
it's a Linux kernel system  
for tracking TCP / UDP  
connections.  
It's a kernel module  
called `nf_conntrack`

## conntrack is used for:

- NAT (in a router!)
- firewalls (eg only allow outbound connections)
- You control it with iptables rules.

conntrack has a table  
of every connection  
Each entry contains:

- src + dest IP
- src + dest ports
- the connection state  
(eg TIME\_WAIT)

## how to enable conntrack

enable:

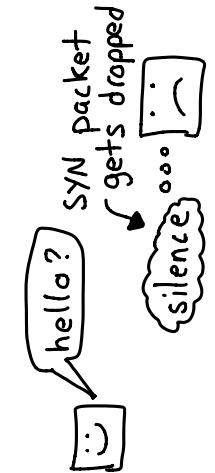
`sudo modprobe nf_conntrack`

check if it's enabled:

`lsmod | grep conntrack`

change table size with the sysctl  
`'net.netfilter.nf_conntrack_max'`

if the conntrack table  
gets full, no new  
connections can start



moral: be careful about  
enabling conntrack!

why are connections  
mysteriously failing?  
maybe the conntrack  
table is full!

# Table of contents ❤

dig.....4	tcpdump.....10-11	ip.....18
ping.....5	tshark.....12	ss / netstat.....19
curl.....6	ngrep.....13	iptables.....20
nmap.....7	openssl.....14	tc.....21
netcat.....8	mitmproxy.....15	conntrack.....22
socat.....9	misc tools.....16	ethtool.....23
ssh.....17		

# dig

## dig makes DNS queries!

\$ dig google.com  
answers have 5 parts:

query: google.com  
TTL: 22  
class: IN (for "internet")  
record type: A  
record value: 172.217.13.110

## dig +trace domain

traces how the domain gets resolved, starting at the root nameservers if you just updated DNS, dig +trace should show the new record

# tc

## make your internet slow

sudo tc qdisc add dev w1p3s0 root netem delay 500ms ← delay packets by 500ms  
and fast again:  
sudo tc qdisc del dev w1p3s0 root netem

## make your brother's internet slow

Have a Linux router? You can configure tc on it to make your brother's internet slower than yours  
google: "tc QoS" for a start

**netem rules**  
netem ("network emulator") is a part of tc that lets you:   
drop, duplicate, delay, corrupt packets. See the man page:  
\$ man netem

21

## dig @ 8.8.8.8 domain

Google DNS server  
dig @server lets you pick which DNS server to query! Useful when your system DNS is misbehaving :)

## dig -x 172.217.13.174

makes a reverse DNS query - find which domain resolves to an IP! Same as dig ptr 174.13.217.172.in-addr.apa

## dig +short domain

Usually dig prints lots of output! With +short it just prints the DNS record

tc  
is for "traffic control"  
packets!  
stop / slow down/  
go the other way!  
great for simulating  
network problems!

## show current tc settings

tc qdisc show  
tc class show dev DEV  
tc filter show dev DEV

tc can do 10 million more things! This is just the beginning!

# iptables

20

iptables lets you create rules to match network packets and accept/drop/modify them It's used for **firewalls** and **NAT**

tables have chains  
chains have rules

tables: filter, nat, mangle, raw, security  
chains: INPUT, FORWARD, PREROUTING, etc  
rules: like `-s 10.0.0.0/8 -j DROP`

**-j TARGET**

Every iptables rule has a **target** (what to do with matching packets). Options:  
→ ACCEPT / DROP / RETURN  
→ the name of an iptables chain  
→ an extension (man iptables-extensions)  
Popular: DNAT, LOG, MASQUERADE

**iptables-save**  
This prints out all iptables rules. You can restore them with iptables-restore but it's also the easiest way to view all rules!

filter: INPUT/OUTPUT/ FORWARD  
mangle: INPUT/OUTPUT/ FORWARD/ PREROUTING/ POSTROUTING  
nat: OUTPUT/PREROUTING/POSTROUTING  
It helps to learn when packets get processed by a given table/chain (eg filter/OUTPUT = all locally generated packets)

**you can match lots of packet attributes**  
-s: src ip      -p: tcp/udp  
-d: dst ip      -i: network interface  
-m: lots of things!  
(bpf rules! cgroups! ICMP type!  
cpu! conntrack state! more!)  
For more, run:  
\$ man iptables-extensions

## Ding & traceroute

ping checks if you can reach a host and its latency  
\$ ping health.gov.au  
output:  
... time=253ms...  
Australia is 17,000 km from me at the speed of light it's still far!

Ping works by sending an ICMP packet and waiting for a reply  
to: health.gov.au  
ping — hello!  
I'm here! — health.gov.au

**myth:** if a host doesn't reply to ping, that means it's down  
Some hosts never respond to ICMP packets. This is why traceroute shows ... sometimes.  
— hello!  
ping  
not listening!! host

traceroute tells you the path a packet takes to get to a destination

me → NYC → Sacramento → my ISP → Australia

**example traceroute**  
\$ traceroute health.gov.au  
1: 192.168.1.1 3ms ← router  
2: ...yul.ebox.ca 12 ms → ISP  
8: NYC4.ALTER.NET 24 ms } crossing the US  
9: SAC1.ALTER.NET 97 ms } time taken  
16: health.gov.au 253ms ← here the packet crossed the USA!  
from NYC → Sacramento!

**mtr**  
like traceroute, but nicer output! try it!

look up how traceroute works (using TTLs!) it's simple + cool!  


# Curl

## curl

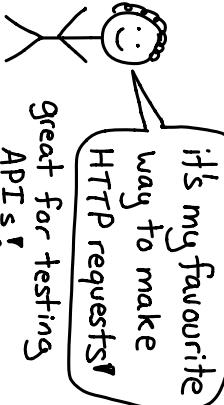
-H  
is for header

good for POST requests to JSON APIs:

-H "Content-Type: application/json"

allow compressed response:

-H "Accept-Encoding: gzip"

  
\$ curl wizardzines.com

-i  
show response headers

-I  
show only response headers  
(makes a HEAD request)

-X POST  
Send a POST request instead  
of a GET (-X PUT etc works too)

-V  
follow 3xx redirects

-k  
show request headers & more

insecure: don't verify  
SSL certificates

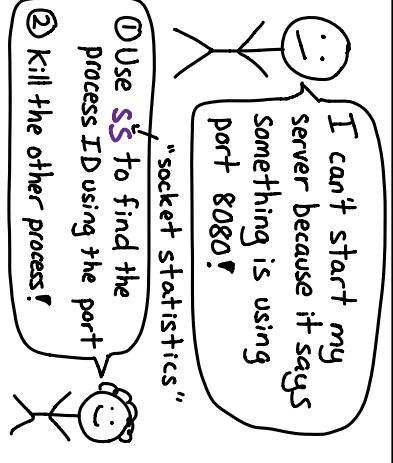
--connect-to ::IP  
send request to IP instead.  
use before changing DNS to a new IP  
or hostname

\* copy as curl \*

Have something in your browser you want to download from the command line?  
In Firefox / Chrome / Safari:  
→ Network tab  
→ right click on the request  
→ copy as curl  
(can have sensitive info in cookies!)

--data  
to POST data!

--data '{ "name": "julia" }'  
--data @filename.json  
@ reads the data to send from a file



# SS

## \* tuna, please! \*

\$ ss -tunap1

the 'a' here doesn't do anything

This is my favourite way to use ss! It shows all the running servers

\*-i \*-m \*

-n  
use numeric ports (80 not http)

-P

show PIDs using the socket

TONS of information

listening or connections?  
non-listening/  
established

which sockets ss shows

which protocols?

default: connections

default: all

-l : listening

-t : TCP

-u : UDP

-a : both

-X : unix domain sockets

## netstat

netstat -tunap1 and ss -tunap1  
do the same thing

netstat is older and more complicated. If you're learning now I'd recommend ss!

# IP

18

## ip

Linux only  
lets you view + change network configuration.  
ip OBJECT COMMAND  
↑  
addr, link add, show, delete, etc  
neigh, etc  
Here are some ways to use it!

## ip addr list

shows ip addresses of your devices. Look for something like this:  
2: eth0:  
link/ether 3c:97...  
inet 192.168.1.170/24

## change your MAC address

good for cafes with time limits 😊

```
$ ip link set wlan0 down  
$ ip link set eth0 address  
3c:a9:f4:d1:00:32  
$ ip link set wlan0 up  
$ service network-manager  
restart ← or whatever you use
```

## ip link

network devices! (like eth0)

## ip neigh

view/edit the ARP table

## ip xfrm

is for IPsec

## ip route list

displays the route table.  
default via 192.168.1.1  
169.240.0.0/16 dev docker0  
...

to see all route tables:  
ip route list table all

## ip route get IP

what route will packets with \$IP take?

## --color

pretty colourful output!

## --brief

show a summary

7

# nmap

## nmap lets you explore a network

which ports are open?  
what hosts are up?

security people use it a lot!

## find which hosts are up

```
$ nmap -sn 192.168.1.0/24  
my home network  
-sn means "ping scan".  
(not -s + -n, it's -sn)
```

just finds hosts by  
pinging every one,  
doesn't port scan

## fast port scan

```
$ nmap -ss -F 192.168.1.0/24  
just sends a SYN packet to  
check if each port is open.  
I found out which ports my  
printer has open!  
80 http  
443 https  
513 printer  
631 ipp  
910 jetdirect
```

## -F or -T5

scan less ports: just the most common ones

## -T4 or -T5

scan faster by timing out more quickly

## aggressive scan

```
$ nmap -v -A scanme.nmap.org  
port, server version, even OS
```

-Pn  
skip doing a ping scan and assume every host is up. good if hosts block ping (lgtm)

## check TLS version ↗ and ciphers

check if your server still supports old TLS versions

```
$ nmap  
--script ssl-enum-ciphers  
-p 443 wizardzines.com  
list all scripts with:  
$ nmap --script-help '*'
```

# netcat

**nc**

lets you create TCP (or UDP) connections from the command line

I hand wrote this HTTP request for you! ☺

start a server! this listens on PORT and prints everything received

network connection → nc → stdout

make HTTP requests by hand

```
printf 'GET / HTTP/\nHost: example.com\r\n\r\n' | nc example.com 80
```

type in any weird HTTP request you want! ☺

be a client! opens a TCP connection to IP:PORT  
(to send UDP use -u)

stdin → nc → network connection

send files

want to send a 100 GB file to someone on the same wifi network? easy!

```
receiver: nc -l 8080 > file
sender: cat file | nc YOUR_IP 8080
```

I ❤️ this trick!  
It works even if you're disconnected from the internet!

# Ssh

★ port forwarding ★

```
ssh user@host.com -NfL
3333:localhost:8888
```

local port → remote port  
lets you view a remote server that's not on the internet in your browser.

.ssh/config

Lets you set, per host:

- username to use
- SSH key to use
- an alias!

mosh

remembers your SSH key passphrase so you don't have to keep typing it

ssh-agent

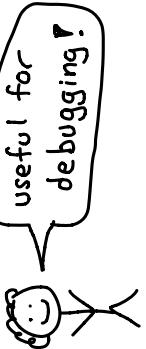
ssh alternative: keeps the connection open if you disconnect + reconnect later

## miscellaneous networking tools

<b>stunnel</b> make a SSL proxy for an insecure server	<b>rsync</b> sync files over SSH or locally	<b>whois</b> is this domain registered?	<b>zenmap</b> GUI for nmap	<b>sysctl</b> configure Linux Kernel's network stack
<b>hping3</b> make any TCP packet	<b>lsof</b> what ports are being used?	<b>ipcalc</b> easily see what 13.21.2.3/25 means	<b>p0f</b> identify OS of hosts connecting to you	<b>ab / iperf</b> benchmarking tools
<b>wget</b> download files	<b>httpie</b> like curl but friendlier	<b>python3 -m http.server</b> serve files from a directory	<b>openvpn wireguard</b> VPNs	<b>links</b> a browser in your terminal
<b>aria2c</b> a fancier wget	<b>ftop / netlog / ntop / iptraf / nload</b> see what's using bandwidth	<b>nftables</b> new version of iptables	<b>tcpflow</b> capture and assemble TCP streams	<b>telnet</b> can help debug text network protocols

## q

## socat

<b>order doesn't matter</b> socat THING1 THING2 is the same as socat THING2 THING1	<b>-V</b> write all transferred data to stderr 
<b>socat supports</b> tcp sockets unix domain sockets SSL sockets pipes files processes UDP sockets ... and MORE!	<b>proxy from local HTTP port to remote server</b> socat TCP-LISTEN:1337 TCP:domain.com:80
<b>socat lets you proxy</b> basically any 2 things the basic syntax: <b>socat THING1 THING2</b>	<b>expose a unix domain socket on port 1337</b> socat TCP-LISTEN:1337 UNIX-CONNECT :/path

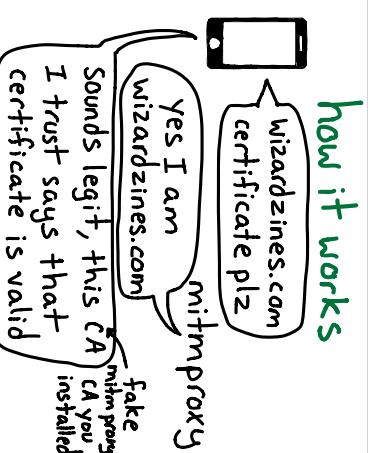
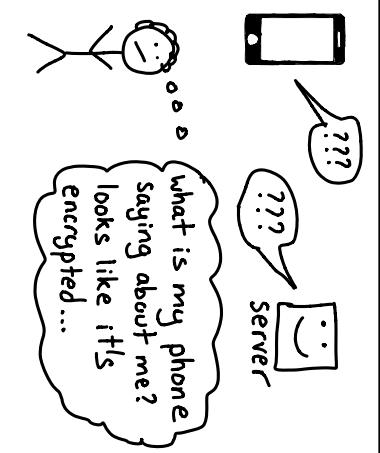
# tcpdump

tcpdump lets you view network packets being sent + received

it's not the easiest to use but it's usually installed ☺

## -w file.pcap

Write packets to a file for later analysis with tcpdump / tshark / wireshark / another tool  
pcap is for "packet capture"



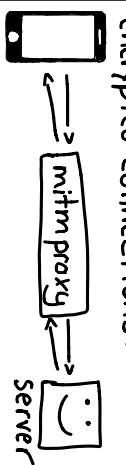
don't try to resolve IP addresses / ports to DNS / port names. makes it run faster.

## -A

print packet contents, not just headers. Nice if you want to quickly see what a few packets contain.

# mitmproxy

mitmproxy can proxy connections from your laptop / phone and let you see the contents. It even works with encrypted connections!



some apps pin a cert makes mitmproxy not work, look up "trust killer" to get around that

## script it in Python

modify requests / responses arbitrarily

## -r

IP addresses / ports to DNS / port names. makes it run faster.

to capture packets on

Which network interface to capture packets on

I often use "-i any" to make sure I'm not missing any packets!

## -c 100000

Only capture a limited count of packets.

I use it with -w so I don't accidentally fill up my disk!

## how you use it

- ① install mitmproxy root CA on your laptop / phone
- ② run mitmproxy web UI on computer
- ③ tell the program / phone to proxy through mitmproxy

15

## other similar tools

(not all free though)

- charles proxy
- burp suite
- fiddler

# OpenSSL

openssl is a tool for doing \*SSL things\*

aka TLS

*Create CSRs*

*Inspect certificates*

*Sign certificates*

It uses the OpenSSL library (or Libressl)

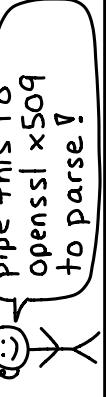
## inspect a certificate

```
$ openssl x509 -in FILE.crt -noout -text
```

this works for files ending in .crt or .pem! Try it out: you probably have certs in /usr/share/ca-certificates

## look at a website's certificate

```
$ openssl s_client -showcerts -connect google.com:443
```



please upload a CSR certificate authority

a WHAT?!

to get a SSL cert for your website, you need to make a file called a "certificate signing request".

## make a CSR

```
$ openssl req -new -sha256 -key FILE.key -out FILE.csr
```

make one of these with \$ openssl genrsa

# BPF cheat sheet

## Berkeley Packet Filter

is a small language you can use to filter which packets tcpdump and ngrep capture.

Use it like this:

```
$ tcpdump [your bpf here]  
$ ngrep [your bpf here]
```

## and / or / not

host 127.0.0.1 and port 80  
(port 53 or port 99) and  
not host 127.0.0.1

## port

```
src port 53  
port 80  
again, same as "src or dst port"
```

## less / greater

```
Packet length!  
less 80  
greater 200
```

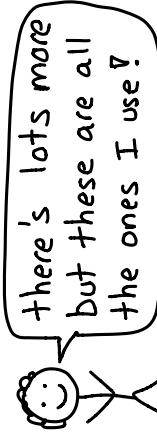
## PROTOCOL [INDEX]

filter based on a specific byte in a packet

IP packets with options:  
ip[0] & "0XF == 5  
DNS SERVFAIL responses:  
udp[11] & 0XF > 0  
SYN packets:  
tcp[tcpflags] == tcp-syn

## tcp / udp / icmp

IP4->IP / IP6  
only show packets using that protocol



# tshark

♥ Wireshark ♥ is an amazing graphical packet analysis tool  
 tshark is the command line version of Wireshark  
 it can do 100x more things than tcpdump ♥

## -T FORMAT

Output format. My favourites:  
 \* json } for these you can specify which fields you want with -e  
 \* fields: csv/tsv } which fields you want with -e  
 \* text: default summary

-Y  
 filter which packets are captured  
 tshark -Y  
 'http.request.method  
 == "GET"  
 ↗ uses Wireshark's SUPER POWERFUL filter language

## -e

Which fields to output. Ex:  
 \$ tshark -T fields  
 -e http.request.method  
 -e http.request.uri  
 -e ip.dst ↗ supports WAY more protocols than HTTP  
 GET /foo 92.183.216.34  
 POST /bar 10.23.38.132

-d  
 is for "decode as"  
 tells tshark what protocol to interpret a port as  
 Example: 8888 is often HTTP!  
 \$ tshark  
 -d tcp.port==8888,http

## -r file.pcap

analyze packets from a file instead of the network

-W ↗ same as tcpdump's Write captured packets to a file. If -w file.pcap has permission issues, try:  
 tshark -w - > file.pcap

# ngrep

## like grep for your network

\$ sudo ngrep GET will find every plaintext HTTP GET request

## -d

is for device  
 which network interface to use. same as tcpdump's {-i} (+try -d any'!)

## -W byline

prints line breaks as line breaks, not "\n".  
 Nice when looking at HTTP requests

## -I file.pcap

-O file.pcap  
 read/write packets from/to a pcap file

## ngrep syntax

\$ ngrep [options] [regular expression] [BPF filter]  
 ↗ same format as tcpdump uses!

